

Programmierkonzepte in der Physikalischen Chemie

18. Tests

Albert-Ludwigs-Universität Freiburg



**UNI
FREIBURG**

Dr. Till Biskup

Institut für Physikalische Chemie
Albert-Ludwigs-Universität Freiburg
Wintersemester 2018/19



- 🔑 Überprüfung ist ein Kernaspekt sowohl von Wissenschaft als auch von moderner Softwareentwicklung.
- 🔑 Überprüfung wissenschaftlicher Software ist entscheidend, weil sie den Erkenntnisgewinn selbst betrifft.
- 🔑 Der größte Teil wissenschaftlicher Software ist unzureichend oder gar nicht getestet.
- 🔑 Tests können nur die Existenz von Fehlern beweisen, nicht deren Abwesenheit.
- 🔑 Tests sollten so früh wie möglich, so oft wie möglich und so automatisiert wie möglich durchgeführt werden.

Motivation: zwingende Voraussetzung für Wissenschaftlichkeit

Probleme mit der Überprüfung von Software

Arten von Tests

Strategien für die Überprüfung von Software

“ *Testing is considered a core principle of scientific software because its impact is at the heart of knowledge generation.*

– Scopatz und Huff

- ▶ Wissenschaftlichkeit erfordert Überprüfbarkeit.
 - unabhängige Überprüfung und ggf. Wiederholung
- ▶ Software ist oft integraler Bestandteil von Forschung.
 - Auswertungen sind i.d.R. nichttrivial.
 - Eine Überprüfung auf Korrektheit ist ebenso nichttrivial.
- ▶ Wissenschaftliche Software muss verlässlich sein.
 - Erkenntnisse und Fortschritt gründen auf Ergebnissen, die mithilfe von Software gewonnen werden.

“ *We would like to think that scientists are rigorous enough to realize the importance of testing [...]*

The truth of the matter is that most scientists are poorly equipped to truly test their code.

The average blog or image-sharing website is better tested than most scientific software.

– Scopatz und Huff

- ▶ Gründe: mangelndes Bewusstsein und Wissen
 - Programmierung hat einen viel zu geringen Stellenwert.
 - Wissenschaftler sind selten ausgebildete Programmierer.

“ *If experimentalists don't calibrate their equipment, check their reagents' purity, and take careful notes, what they're doing isn't considered science. In contrast, computationalists don't even learn how to assess their software's quality in any systematic way, and very few would be able to recreate and rerun the programs they used to produce last year's papers. As a result, **most computational science is irreproducible and of unknown quality.***

– Greg Wilson

- Sorgfältige Tests wissenschaftlicher Software sind eine Frage der *Professionalität als Wissenschaftler!*

- ▶ Tests sind destruktiv.
 - Ein erfolgreicher Test ist ein Test, der zu einem Fehler führt.
 - widerspricht dem Entwicklergedanken: etwas erschaffen
- ▶ Tests erfordern die Erwartung, Fehler zu finden.
 - Wer keine Fehler finden will, wird sie oft übersehen.
 - Entwicklern fehlt häufig die notwendige Objektivität.
- ▶ Tests sagen nichts über Fehlerfreiheit.
 - Tests können nur (bislang unbekannte) Fehler finden.
 - Die Abwesenheit von Fehlern ist allgemein nicht beweisbar.
- ▶ Tests selbst erhöhen nicht die Softwarequalität.
 - Testergebnisse müssen ernst genommen und Fehlerursachen behoben werden.

- ▶ Programme sind meist schwer zu testen.
 - Testbarkeit erfordert Modularität und Unabhängigkeit.
 - Entwicklung fokussiert meist nicht auf Testbarkeit.
- ▶ Tests haben einen viel zu geringen Stellenwert.
 - Tests fallen oft dem Zeitmangel zum Opfer.
 - Das Bewusstsein für die Bedeutung verlässlicher Software gerade in der Wissenschaft fehlt überwiegend.
- ▶ Es fehlt das Wissen für effektive Tests.
 - Tests zu schreiben ist nicht trivial.
- ▶ Testen ist langweilig und fehleranfällig.
 - Automatisierung ist das Gebot der Stunde.
 - Tests sollten automatisiert *ausgewertet* werden.

Vollständige Tests sind meist undurchführbar

Entscheidend ist die geschickte Wahl aussagekräftiger Testfälle.

Satz

Der mögliche Parameterraum jedes nichttrivialen Programms ist zu groß für vollständige Tests.

- ▶ Beispiel: Funktion zum Speichern von Adressen
 - Name/Adresse: 20 Buchstaben, Telefonnummer: 10 Ziffern
 - Möglichkeiten: $26^{20} \times 26^{20} \times 10^{10} \approx 10^{66}$
 - Vergleich: vermutlich ca. 10^{80} Atome im Universum
- ▶ Abhilfe: intelligente Wahl der Testfälle
 - charakteristische Testfälle
 - erfordert Vertrautheit mit der Problemstellung und Kenntnis über Spezialfälle

Satz

Fehlerfreiheit von Programmen ist im Allgemeinen weder erreichbar noch beweisbar.

- ▶ Fehlerfreiheit ist kein sinnvolles Kriterium.
 - Aber: Unerreichbarkeit ist kein Argument gegen Tests.
- ▶ Vertrauenswürdigkeit von Code ist entscheidend.
 - Systematische Tests können Vertrauen schaffen.
 - Entscheidend ist nicht die Zahl von Tests, sondern ihre Abdeckung unterschiedlicher Fälle.
 - Abdeckung aller Zustände ist entscheidend.
 - Abdeckung aller Codezeilen ist ein schlechtes Maß.

These

Numerische Simulationen und Rechnungen lassen sich oft nicht mit bekannten Lösungen überprüfen.

- ▶ Erkenntnisgewinn ist Ziel der Wissenschaft.
 - Oft gibt es keine bekannten Lösungen.
 - Software ist oft Werkzeug des Erkenntnisgewinns.
- ▶ Lösungsansatz: Vertrauen in die Einzelteile
 - Elementaroperationen müssen zuverlässig korrekt sein.
 - Voraussetzung: modularer, lesbarer, testbarer Code
- ▶ Lösungsansatz: unabhängige Implementierung
 - Implementierungsfehler sollten statistisch unabhängig sein.

Kriterien zur Einteilung von Tests

Tests lassen sich nach diversen Kriterien unterscheiden.

- ▶ Durchführende
 - Entwickler, Tester, Kunden, Anwender
- ▶ Größe der zu testenden Einheit
 - Unittests, Integrationstests, Systemtests
- ▶ Kenntnis der Implementierung
 - White-Box-Tests, Black-Box-Tests
- ▶ Reproduzierbarkeit
 - reproduzierbar, abhängig von äußeren Faktoren
- ▶ Art der Durchführung
 - manuell, automatisiert
- ▶ Zielstellung
 - Regression, Validierung, Entwicklung

- ▶ Entwickler sind für die Tests verantwortlich.
 - Entwickler sind in vielen Fällen selbst Nutzer.
 - Entwickler sind meist Einzelkämpfer.
 - Implementierung ist i.d.R. bekannt.
- ▶ Wissenschaftliche Software dient dem Erkenntnisgewinn.
 - korrekte Lösungen für komplexe Probleme nicht bekannt
 - Ausweg: Überprüfung der einzelnen Codebestandteile
- ▶ Tests erfordern Kompetenzen auf zwei Gebieten.
 - Gebiete: Programmierung und Wissenschaft
 - Implementierungen sind immer fehleranfällig.
 - Korrekte Implementierungen garantieren nicht wissenschaftliche Korrektheit.

▶ Unittest

- Test eines Codeblocks in Isolation
- Getestete Codeblöcke sind in der Regel klein.
- Verhalten des Codeblocks ist eindeutig definierbar.

▶ Integrationstest

- Test des Zusammenspiels mehrerer Codeblöcke
- Einzelne Codeblöcke wurden vorher getestet (Unittests).
- Test nicht mehr (notwendigerweise) in Isolation

▶ Regressionstest

- Test auf Übereinstimmung mit früherer Funktionalität
- relevant bei Veränderungen im Code
- stellt sicher, dass korrekter Code korrekt bleibt
- Voraussetzung: Korrektheit der Tests

“ *The single most important rule of testing is to do it.*

– Kernighan und Pike

▶ früh testen

- erleichtert das Eingrenzen von Fehlern
- verhindert Fehlentwicklungen

▶ oft testen

- mindestens einmal pro (Arbeits-)Tag
- Fehler beheben, bevor weiter entwickelt wird

▶ automatisiert testen

- Voraussetzung für häufiges Testen
- verringert die Fehlerquote beim Testen

- ▶ systematische Tests
 - jede Zeile im Produktivcode abtesten
 - jede Bedingung einzeln abtesten
 - Ziel: alle möglichen Zweige des Programms abtesten
- ▶ Kontrollfluss und Datenfluss testen
 - drei Zustände für Daten: definiert, benutzt, freigegeben
 - Atypische Folgen von Zuständen deuten auf Fehler hin.
- ▶ Grenzen analysieren
 - Grenzen von Wertebereichen sind besonders fehleranfällig.
 - besondere Vorsicht bei mehreren abhängigen Parametern
- ▶ Hilfsmittel zum Test in Isolation verwenden
 - Ziel: Test eines einzelnen Codeblocks
 - externe Abhängigkeiten durch Testumgebung auflösen

Wie viele Tests sind ausreichend?

Die McCabe-Metrik liefert die minimale Zahl von Testfällen.

Die McCabe-Metrik als untere Grenze

- ▶ Ziel
 - vollständige Abdeckung der Zweige eines Programms mit der minimalen Anzahl von Tests.
- ▶ Regeln zur Bestimmung der Zahl notwendiger Tests
 - 1 – gerader Weg durch das Programm
 - +1 für jedes `if`, `while`, `repeat`, `for`, `and`, `or`
 - +1 für jedes `case`
 - +1 wenn kein default-Zweig im `switch/case`
- ▶ Grenzen
 - deckt lediglich alle Zweige eines Programms ab
 - Überprüfung der Wertebereiche von Parametern erfordert zusätzliche Tests.

Tests selbst sind nicht fehlerfrei

Überprüfung der Tests führte zu einer unendlichen Regression.

Problem

- ▶ Tests sind Code und deshalb genauso fehleranfällig.
 - Test der Tests ist keine Lösung.

mögliche Lösungsstrategien

- ▶ Testcode ernst nehmen
 - Testcode ist genauso wichtig wie Produktivcode.
 - Testcode sollte genauso sorgfältig geschrieben werden.
 - Alle genannten Aspekte von „Sauberem Code“ anwenden.
- ▶ Testfälle frühzeitig planen
 - hilfreich zur Vermeidung falscher Annahmen
- ▶ Testcode nicht wegwerfen
 - hilft dabei, Testcode ernst zu nehmen

- ▶ Testcode nicht wegwerfen
 - Tests sind ein wichtiger Bestandteil der Entwicklung.
 - Tests sollten immer wieder ausgeführt werden können.
- ▶ Testcode sauber schreiben
 - Lesbarkeit ist mindestens so wichtig wie bei Produktivcode.
 - Tests sind ausführbare Spezifikation und Dokumentation.
- ▶ Testcode zentral lagern
 - eigenes Verzeichnis oder direkt neben dem Produktivcode
 - Konsistente Benennung erhöht die Übersichtlichkeit.
- ▶ Testcode von Produktivcode trennen
 - Testcode dient *nicht* dem Debugging.
 - Produktivcode sollte *von außen* testbar sein.



- 🔑 Überprüfung ist ein Kernaspekt sowohl von Wissenschaft als auch von moderner Softwareentwicklung.
- 🔑 Überprüfung wissenschaftlicher Software ist entscheidend, weil sie den Erkenntnisgewinn selbst betrifft.
- 🔑 Der größte Teil wissenschaftlicher Software ist unzureichend oder gar nicht getestet.
- 🔑 Tests können nur die Existenz von Fehlern beweisen, nicht deren Abwesenheit.
- 🔑 Tests sollten so früh wie möglich, so oft wie möglich und so automatisiert wie möglich durchgeführt werden.